National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

DREA CR/97/460

FIR NEURAL NETWORK SOFTWARE PAKAGE USER'S MANUAL

Maryhelen Stevenson — Normand Arsenault

Center for Research in Engineering and Applied Science
University of New Brunswick
P.O.Box 440, Fredricton, N.B., Canada
E3B 5A3

CONTRACTOR REPORT

Prepared for

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

106-03682/

THIS IS AN UNEDITED REPORT ON SCIENTIFIC OR TECHNICAL WORK CONTRACTED BY THE DEFENCE RESEARCH ESTABLISHMENT ATLANTIC OF THE RESEARCH AND DEVELOPMENT BRANCH OF THE DEPARTMENT OF NATIONAL DEFENCE, CANADA.

THE CONTENTS OF THE REPORT ARE THE RESPONSIBILITY OF THE CONTRACTOR, AND DO NOT NECESSARILY REFLECT THE OFFICIAL POLICIES OF THE DEPARTMENT OF NATIONAL DEFENCE.

PLEASE DIRECT ENQUIRIES TO:

THE DIRECTOR GENERAL
DEFENCE RESEARCH ESTABLISHMENT ATLANTIC
P.O. BOX 1012
DARTMOUTH, NOVA SCOTIA, CANADA
B2Y 3Z7

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

DREA CR/97/460

FIR NEURAL NETWORK SOFTWARE PACKAGE USER'S MANUAL

Maryhelen Stevenson — Normand Arsenault

Center for Research in Engineering and Applied Science
University of New Brunswick
P.O.Box 440, Fredricton, N.B., Canada
E3B 5A3

Accepted by : Jnc Maksym
J.N. Maksym - Contract Scientific Authority

W7707-5-2682/01-OSC
Contract Number

April 1995

CONTRACTOR REPORT

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

Abstract

User's manual (revised edition, April 1995) for Finite Impulse Response Neural Network software written in the C programming language under the contract. The revised version includes the operation of a Graphic User Interface (GUI), and is updated to reflect all the changes in the use and operation of the FIR Neural Network and the GUI software packages.

Résumé

Manuel de l'utilisateur (édition révisée, avril 1995) du logiciel de réseau neuronal à réponse impulsionnelle finie, écrit en langage de programmation C aux termes du marché. La version révisée inclus le fonctionnement de l'interface utilisateur graphique (GUI) et est mise à jour pour refléter toutes les modifications visant l'utilisation et le fonctionnement du réseau neuronal FIR et des logiciels GUI.

TABLE OF CONTENTS

Abstract ii
Table of Contents iii

Preface v

Preface (Revised Edition) v

1 FIR Neural Network Software Components **1**

1.1 Makefile 1

1.2 fir.h 1

1.3 fir_nn.c 2

1.3.1 read user options from the input file named user_options 2

1.3.2 build FIR neural network according to user–specified architecture ... 2

1.3.3 read remainder of user options from input file named user_options ... 2

1.3.4 initialize the pattern length, T, as deduced from network architecture . 2

1.3.5 initialize desired–response sequence to be used for the true–class output node 3

1.3.6 initialize error–scaling sequence 3

1.3.7 read in training set and testing set 3

1.3.8 train the network 4

1.3.9 evaluate mean square error and classification rate on test and training sets (if requested by the user) 5

1.3.10 dump weights to a user–specified file (if requested by the user) 5

1.3.11 operation mode 5

1.3.12 Proc_Sig 5

1.4 construct_net.c 6

1.4.1 note on connectivity of penultimate–layer nodes to final–layer nodes . 6

1.5 read_set.c 6

1.6 forward_pass.c 7

1.7 find_neg_errs.c 7

1.8 backward_pass.c 7

1.9 update_weights.c 7

1.10 test_patterns.c 8

1.11 mse_patterns.c 8

1.12 clear_act.c 8

1.13 clear_net.c 9

1.14 dump_weights.c 9

1.15 save_pat_output.c 9
1.16 free_set.c 9
1.17 read_cont_set.c 9
1.18 free_cont_set.c 9
1.19 save_iter_output.c 10
1.20 test_continuous.c 10

2 Contents and Organization of Input and Output Files 11

2.1 The user–options file, user_options 11
2.1.1 warning regarding the length of user–specified filenames 13
2.2 The network–parameter file 13
2.2.1 warnings regarding specification of network architecture 15
2.3 data–set files 15
2.3.1 warning regarding the reading in of the data–files 16
2.3.2 warning regarding the class numbers 16
2.4 weight files 16
2.4.1 note to user regarding nonadaptable weights 17
2.5 classification–result files 17
2.6 MSE–result files 18
2.7 operation mode classification result file 18
2.8 layer output files 18

3 Using xfir 20

3.1 Options 20
3.1.1 Edit user_options 20
3.1.2 Edit Net Parameters 20
3.1.3 Begin Simulation 21
3.1.4 End Simulation 21
3.2 Display 21
3.2.1 Network Outputs 22
3.2.2 Class Decision Summary 22
3.3 fir_nn Status Display 23
3.4 Quit 23

References 24

PREFACE

This document is intended for the user's of the Finite Impulse Response (FIR) Neural Network software package which was developed by Maryhelen Stevenson and Normand Arsenault in the Department of Electrical Engineering at the University of New Brunswick. The package simulates a FIR Neural Network classifier; it makes use of the Temporal Backpropagation training algorithm. An explanation of the theory behind the FIR neural network architecture and the temporal backpropagation training algorithm can be found in [2].

The software package was developed in the C programming language; it has been compiled and tested on a UNIX–based SUN SPARC station. Experimental results which were obtained using this package can be found in [1].

The purpose of this manual is to describe the organization and operation of the software package. The manual is organized in two main sections. The first section describes the various routines which make up the software package; the second section describes the input and output files.

PREFACE (REVISED EDITION)

The changes incorporated in the FIR Neural Network software package were made to reflect the revised format of the user options file and to allow the FIR software to operate in conjunction with the newly developed Graphical User Interface (GUI) whose operation is described in section 3. The updated user options file contains all the previous options with some additional features and leaves room for future expansions. The FIR software package and GUI software package are separated by a clean interface, thus allowing the FIR software to be operated independently or in conjunction with the GUI. A short chapter was included to describe the use and operation of the GUI. This manual was therefore updated to reflect all the changes in use and operation of the FIR Neural Network and GUI software packages.

1 FIR Neural Network Software Components

The source code for the FIR Neural Network software package is comprised of the following files. Each file contains one function. The main function is contained in the file fir_nn.c; the header file fir.h defines the constants and structure types used by the various functions.

1. Makefile
2. fir.h
3. fir_nn.c
4. construct_net.c
5. read_set.c
6. forward_pass.c
7. find_neg_errs.c
8. backward_pass.c
9. update_weights.c
10. test_patterns.c (was classify.c)
11. mse_patterns.c (was mn_squ_err.c)
12. clear_act.c
13. clear_net.c
14. dump_weights.c
15. save_pat_output.c (was save_output.c)
16. free_set.c
17. read_cont_set.c
18. free_cont_set.c
19. save_iter_output.c
20. test_continuous.c

A brief description of the files listed above is given below.

1.1 Makefile

The Makefile contains the commands which will compile the source files, link the resulting object files, and create the executable file, fir_nn. To execute the Makefile, the user need only type *make* within the directory which houses the source files listed above.

1.2 fir.h

This header file defines the constants and the data structures that will be used to simulate the FIR Neural Network and to store the data sets. The constants which are assigned val-

ues in fir.h include MAX_IO_NODES, BIAS_IN, MAXFILENAME, MAXINPUTS, MAXLENGTH, MAXLAYERS, NUMMODES, MAXCHAR, TRUE, and FALSE. The structure types which are defined include: PairType, SetType, ArrayType, VectorType, FilterType, NodeType and NetType.

1.3 fir_nn.c

This file contains the main function of the FIR Neural Network software package. The general flow of this function is summarized by the subheadings below.

1.3.1 read user options from the input file named *user_options*

The user options required to build the FIR Neural Network are read[1] in from a file named *user_options* which should reside in the same directory as the FIR software package. The organization and contents of the file named *user_options* are discussed in a section 2.1.

1.3.2 build FIR neural network according to user-specified architecture

One of the items read from the user-options file is the name of the file which contains the specifications of the desired FIR network architecture. For convenience, we will refer to this file as the *network-parameter file*. The main function calls Construct_Net (see section 1.4) to build the network as specified by the contents of the network-parameter file. The organization and contents of the network-parameter file are discussed in section 2.2.

1.3.3 read remainder of user options from input file named *user_options*

The revised user options file contains a line (see item number 29 of Table 1) in which the number of elements is dependant on the size of the network; the remaining user options are read after the FIR neural network is constructed such that the number of elements in the *user_options* file is known.

1.3.4 initialize the pattern length, T, as deduced from network architecture

The length, T, of the input patterns (sometimes called sequences or tokens), is inferred from the specified network architecture. Each pattern in the training set is assumed to be a sequence of vectors. Each vector in the sequence has a number of elements equal to the number of network inputs; each sequence is assumed to have length T, where T is the length of the FIR network's impulse response (one more than the sum of filter orders over all layers in the network).

---

1. Due to the large number of user options, we have found it to be much more convenient to read in the user options from a file rather than to enter them interactively. The interactive approach is often more time consuming, more error prone, and does not allow the process to be run in the background.

1.3.5 initialize desired–response sequence to be used for the true–class output node

Desired–response sequences of length 2T–1 (the number of iterations required to clock an input sequence of length T in and out of the network) must be generated for each output node. All elements of the desired response sequences for the output nodes not associated with the true class of the given input sequence are set to the negative of the desired response sequence for the true class node. The software allows the user to choose between two types of desired response sequences to be applied to the output of the node corresponding to the true class of the given input sequence. The user indicates his/her choice by appropriately setting the value of the desired response flag which is read from the user–options file.

If the user sets desired_resp_flag equal to 0, the desired response for the output node corresponding to the true pattern class will be set to 1 for the length of time that it takes to clock the input pattern in and out of the network (a total of 2T–1 iterations); if the user sets desired_resp_flag equal to 1, the desired response for the same output node will be a gaussian–shaped function (of the iteration number) – it will have a value close to 0 when the first vector of the input pattern is clocked into the network, it will reach its peak value of 1 at the time when the pattern fills the network, and will taper back down to 0 as the pattern is clocked out of the network.

1.3.6 initialize error–scaling sequence

Error–scaling sequences of length 2T–1 (the number of iterations required to clock an input sequence of length T in and out of the network) are used to emphasize errors (between outputs and desired responses) occurring at certain instants of time more heavily than others. Normally, we would like to place more emphasis on the error corresponding to the instant of time for which the entire input sequence fills the network and less emphasis on the error at instants of time when only a portion of the sequence has been clocked into the network.

The software allows the user to choose between three types of error–scaling sequences. The user indicates his/her choice by appropriately setting the value of the error scaling flag which is read from the user–options file. If the user sets ScaleErrsFlag equal to 0, equal emphasis is placed on all errors. If the user sets ScaleErrsFlag equal to 1, all emphasis is placed on the error corresponding to the instant of time when the input sequence fills the network. Finally, if the user sets ScaleErrsFlag equal to 2, most of the emphasis is placed on the instant of time when the input sequence fills the network; the emphasis tapers down to zero (following a gaussian–shape curve) for instants of time corresponding to when only one vector of the input sequence has been clocked into, or has yet to be clocked out of, the network.

1.3.7 read in training set and testing set

The names of the files containing the training and testing sets are user inputs included in the user–options file. The function Read_Set or Read_Cont_Set (see section 1.5 or

1.17) is invoked to read in the training and testing set from the appropriate files. The organization and contents of the files containing the testing and training sets are discussed in section 2.3.

1.3.8 train the network

The network is trained by cycling through the patterns in the training set until the total number of pattern presentations exceeds the constant MaxPat; the value of MaxPat is set by the user in the user–options file. Each pattern is assumed to be a vector–valued sequence of length T. A total of 3T–2 iterations are required to accumulate the weight updates resulting from a pattern of length T. Note that during the first T iterations, the pattern is being clocked into the network; during the next T–1 iterations the pattern is being clocked out of the network and zeros are clocked into the network; the final T–1 iterations are used to finish backpropagating the errors through the network. Depending on the iteration number, one of four sets of instructions are applied as outlined below.

instructions for the first T iterations ($0 \leq$ PatIter $\leq T-1$):
- present next vector of current input sequence to the network
- calculate new outputs for all processing units (Forward_Pass)
- evaluate errors at network output (Find_Neg_Errs)
- propagate scaled errors backward through network (Backward_Pass)
- prescribe weight updates for current iteration (Update_Weights)

instructions for the next T–1 iterations ($T \leq$ PatIter $\leq 2T-2$):
- present vector of zeros as input to the network
- calculate new outputs for all processing units (Forward_Pass)
- evaluate errors at network output (Find_Neg_Errs)
- propagate scaled errors backward through network (Backward_Pass)
- prescribe weight updates for current iteration (Update_Weights)

instructions for the next T–2 iterations ($2T-1 \leq$ PatIter $\leq 3T-4$):
- present vector of zeros as input to the network
- calculate new outputs for all processing units (Forward_Pass)
- propagate vector of zeros backward through network (Backward_Pass)
- prescribe weight updates for current iteration (Update_Weights)

instructions for the final iteration (PatIter $= 3T-3$):
- present vector of zeros as input to the network
- calculate new outputs for all processing units (Forward_Pass)
- propagate vector of zeros backward through network (Backward_Pass)
- set UpdateFlag appropriately
- prescribe weight updates for current iteration (Update_Weights)

Note that the instructions for the final iteration are stated separately so that UpdateFlag can be set appropriately in the case the user has chosen to make pattern–wise updates as opposed to iteration–wise updates. If UpdateFlag is set to 1, the function Update_Weights prescribes weight changes and changes the weights accordingly; otherwise, if UpdateFlag is set to 0, the function Update_Weights simply accumulates the currently prescribed changes but does not actually update the weights. If the variable

PatsPerUpdate (read from the user–options file) is equal to zero, then the weights are updated each iteration; otherwise, the prescribed weight updates are accumulated every iteration but the weights are updated only once every PatsPerUpdate pattern presentations.

1.3.9 evaluate mean square error and classification rate on test and training sets (if requested by the user)

At the end of training (as determined by MaxPat), as well as at the end of each training epoch, a check is made to see whether or not the mean square error and/or classification rate of the training and/or test set is to be evaluated. The function MSE_Patterns (see section 1.11) is invoked to evaluate the mean square error on a data set and the function Test_Patterns or Test_Continuous (see section 1.10 or 1.20) is invoked to evaluate the classification rate on a data set. Frequency of evaluation of the classification rate on the test set is specified by the value of epochs_per_test_class; this value is read from the user–options file. Whenever the number of training epochs (*i.e.*, the number of cycles through the training set) reaches an integral multiple of the variable epochs_per_test_class, the classification rate of the test set is evaluated. The classification rate of the test set is never evaluated if the value of epochs_per_test_class is zero. Similarly, the frequencies of evaluation of the classification rate on the training set, the mean square error on the test set, and the mean square error on the training set, are determined respectively by the user–options: epochs_per_train_class, epochs_per_test_mse, and epochs_per_train_mse.
note: to evaluate the MSE or classification rate on the test and/or training set without training, the user should set MaxPat equal to 0 and the appropriate frequency–of–evaluation variables equal to 1.

1.3.10 dump weights to a user–specified file (if requested by the user)

When training is complete, the value of save_weights (as specified in the user–options file) is checked; if save_weights has a value of 1, the weights are written to a file (the function Dump_Weights, discussed in section 1.14, is invoked for this purpose).
note: The user now has the capability to stop training and store the weights when desired. By pressing "control c" the user generates an interrupt signal which is caught by the FIR Neural Network software which calls the function Proc_Sig (see section 1.3.12).

1.3.11 operation mode

When training has terminated the user may wish to present a third data set, the operation mode data set whose name in contained in the user options file, to the FIR network. With the weights held constant this data set is propagated through the network and the outputs of any layer can be saved to a file for further study.

1.3.12 Proc_Sig

This function is contained in the fir_nn.c file after the main program. When an interrupt (control c) is caught by the signal handler this function is executed. The current weight

values will be saved to a file called *dumped.weights*. When the Dump_Weights function is finished executing the FIR program will terminate.

1.4 construct_net.c

This file contains the function Construct_Net. Construct_Net reads the specifications for the desired network from the user–generated network–parameter file (a pointer to this file, p_file, is passed from the main function). It then allocates space for all the NodeType and FilterType structures needed to build the desired FIR network. In case of a memory allocation problem, the function returns a value of –2 to the main function. If the parameter usedwgts has a value of 1, the function reads in the values of the network's weights from a user–specified weight file (a pointer to this file, p_wgt, is passed from the main function); otherwise, the adaptable weights of the network are assigned random initial values uniformly distributed between –max_init_wt and +max_init_wt, where the value of max_init_wt is read from the network parameter file.

The organization and contents of the network–parameter file are discussed in section 2.2.

1.4.1 note on connectivity of penultimate–layer nodes to final–layer nodes

It should be noted that the FIR network constructed by this software package is set up to act as a time–invariant classifier. The time–invariant operation of the classifier is accomplished by making the weights of the output layer nonadaptable. It is assumed that the network has the same number of nodes in the penultimate layer as it has in the output layer. The weights of the filters which connect corresponding nodes in the penultimate and output layers are all set to a constant which is equal to the reciprocal of the number of filter weights; thus the purpose of these filters is simply to average the past outputs of the associated penultimate–layer node. The weights of all other filters connecting penultimate–layer nodes to final–layer nodes as well as the bias weights of the final–layer units are set to zero.

1.5 read_set.c

This file contains the function Read_Set which is used to allocate the space required to store a pattern grouped data set and to read in the data set from the file specified by the file pointer p_data_file. The data set is stored in the SetType structure pointed to by the parameter p_set. The number of pattern/desired–response pairs to be read in from the file is specified by the num_pairs member of the specified SetType structure. The number of components per pattern is specified by the parameter pat_size. In case of a memory allocation problem, the function returns a value of –2 to the main function.

1.6 forward_pass.c

This file contains the function Forward_Pass. Forward_Pass loops through all nodes in all the layers of the network (starting with layer 0, the inputs, and proceeding to layer L, the output nodes). For each node on each layer, new linear outputs are calculated, the contents of the shift registers storing the past values of the node's activations[2] and derivatives[3] of each node are shifted, and the current activations and derivatives are computed.

1.7 find_neg_errs.c

This file contains the function Find_Neg_Errs. The function Find_Neg_Errs subtracts the current desired output from the actual output for each output node. The vector of negative errors[4] is then stored in a VectorType structure.

1.8 backward_pass.c

This file contains the function Backward_Pass. Backward_Pass propagates the output errors backward through the network to find the deltas for each node of the network (starting with the nodes of the output layer and working backward to the nodes of the first layer). The delta of a given node is a measure of the sensitivity of the sum of squared output errors for the current input sequence with respect to the linear output of the node *delay* iterations ago. Note that the linear output of a node (at a given instant of time) affects the output error for several iterations into the future and thus there is a delay of *delay* iterations before its total effect can be evaluated. This is discussed in more detail in [2].

1.9 update_weights.c

This file contains the function Update_Weights. This function cycles through all nodes in the network except the output-layer nodes; the weights associated with the output-layer nodes are nonadaptable (see section 1.4). For each node, the function prescribes changes to all associated weights and updates the associated weight-change accumulators. If the parameter UpdateFlag (passed from the main function) is equal to 1, the function updates the weights according to the accumulated weight changes and resets the accumulators to zero.

---

2. The activation of a node refers to its nonlinear output.

3. The derivative of a node refers to the derivative of the node's nonlinear function with respect to its argument evaluated at the node's linear output.

4. We refer to these errors as negative errors, because the error is normally defined as (desired – actual).

1.10test_patterns.c

This file contains the function Test_Patterns. This function evaluates the classification rate of the pattern set to which the parameter p_set points. While evaluating the classification rate, the weights of the network are held constant. The results are written to the file specified by the file pointer p_out_file. If the parameter long_cls_version has a value equal to one, the verbose version of the classification results are written to the output file; otherwise, a concise summary of the classification results are written to the output file. The summary of the classification results includes the number of patterns correctly classified, the number of patterns incorrectly classified, and the resulting classification rate. The verbose version of the classification results allows the generation of a confusion matrix; it includes a line of output for every pattern in the set followed by a summary line. In the verbose version, the line of output generated for each pattern includes the class of the pattern as decided by the network, the true class of the pattern, and a one or zero to indicate whether or not the pattern was correctly classified by the network (a one indicates correct classification).

1.11mse_patterns.c

This file contains the function MSE_Patterns. The function MSE_Patterns evaluates a weighted sum of squared errors (over all output units, over the entire duration of each input sequence and over all sequences) for the pattern set to which the parameter p_set points. The weighted sum of squared errors is the cost function which the training procedure attempts to minimize with respect to the patterns in the training set. To evaluate the cost function, each pattern is clocked in and out of the network, while holding the network weights constant; at each instant of time, the error between the desired and actual network output is computed, squared, and weighted by the appropriate component of the Error-Weights array in the case that the user has chosen to invoke the error scaling option. The weighted squared errors are accumulated and then normalized by the number of network outputs, the number of iterations required to clock a pattern in and out of the network, and the number of patterns in the set. The final result is written to the file specified by the file pointer p_out_file.

1.12clear_act.c

This file contains the function Clear_Activations which is used to clear all the past and present values of activation and derivative values. The function is called at the end of each pattern, not including the final pattern, during the classification procedure.

1.13clear_net.c

This file contains the function Clear_Network which is used to clear all the present and past values of activations, derivatives and deltas. This function is called at the end of the classification procedure.

1.14dump_weights.c

This file contains the function Dump_Weights. Dump_Weights writes every weight of the network to the file specified by the file pointer p_wt_file. The format of the weight file is discussed in section 2.4.

1.15save_pat_output.c

This file contains the function Save_Pat_Output which is used to dump the output from any layer of the network to the files specified by the array of file pointers p_layer_out_file[0], p_layer_out_file[1], ... , p_layer_out_file[L]. The outputs are generated while holding the weights of the network fixed and clocking in and out the first number_of_pats patterns contained in the pattern set to which the parameter p_set points. The format of the resulting output file(s) is discussed in section 2.8.

1.16free_set.c

This file contains the function Free_Set which is used to deallocate the space required to store a pattern grouped data set (see section 1.5 for details on how memory is allocated). The memory required to store each PairType structure is released, when all num_pairs patterns have been processed the memory required to store the SetType structure is freed.

1.17read_cont_set.c

This file contains the function Read_Cont_Set. Read_Cont_Set is used to allocate the space required to store a data set consisting of a continual stream of input sequences[5] and to read in the data set specified by the file pointer p_data_file. The number of input sequences and data is stored in the ArrayType structure pointed to by p_data_set. In case of a memory allocation problem, the function returns a value of –2 to the main program.

1.18free_cont_set.c

This file contains the function Free_Cont_Set which is used to deallocate the space required to store a continuous data set (see section 1.17 for details on how this memory is

5. The ability to use continuous data sets, stream of input sequences which have not been segmented into patterns, is a new addition to the FIR package and has not been tested exhaustively. The FIR software is configured such that a continuous format data set may be used to replace the pattern grouped test set.

allocated). Input sequences pointed to by the ArrayType structure p_data_set are freed until all input sequences have been released.

1.19save_iter_output.c

This file contains the function Save_Iter_Output. Save_Iter_Output saves the output of the layers of the network specified by the array of flags: save_layer_flags[0], save_layer_flags[1], ... , save_layer_flags[L]. The output files are specified by the array of file pointers: p_layer_out_file[0], p_layer_out_file[1], ... , p_layer_out_file[L]. The format of the resulting output file(s) and naming convention is discussed in section 2.8.

1.20test_continuous.c

This file contains the function Test_Continuous which saves the output to files specified by the array of file pointers: p_layer_out_file[0], p_layer_out_file[1], ... , p_layer_out_file[L]. This allows the network outputs to be processed external to the FIR software package. This will useful for determining suitable threshold values to use when making classification decisions based on whether a given output is above its associated threshold.

2 Contents and Organization of Input and Output Files

Before executing the FIR neural network package, a number of input files must be created by the user and housed in the same directory as the executable. Essential files include the user–options file and the network–parameter file. To do anything meaningful, one or more data–set files must exist. If the user chooses to read the initial weights in from a file, a weight file must also exist. The only file name which is set by the software is the name of the user–options file which is *user_options*. All other file names will be specified by the user in the user–options file.

Depending on the contents of the user–options file several output files may be created by the program. These include a new–weight file, a testing–set classification result file, a training–set classification result file, a testing–set mean–square–error file,a training–set mean–square–error report file, an operation mode classification result file, and various layer output files.

The contents and organization of the input and output files are described below.

2.1 The user–options file, *user_options*

The content and organization of the user–options file is summarized in Table 1. There is one row of the table for each variable read from the file. The order of the rows is identical to the order in which the variables should appear in the file. The first column of the table indicates the item number, the second column gives a description of the item, the third column states the associated variable name, and the fourth column states the variable type (where cs indicates a character string and int indicates an integer).

Table 1: Content and Organization of User–Options File, *user_options*

no.	description	variable name	type
1	name of network parameter file which contains specifications for desired architecture and training parameters	net_par_file	cs
2	option to read in weights from file 0: assign random weights 1: read in weights from old–weight file	use_saved_weights	int
3	name of file from which initial weights of the network will be read	old_wt_file	cs
4	name of file containing training set	train_file	cs
5	training set file type: 0: pattern grouped	train_set_type	int

Content and Organization of User–Options File, *user_options* (continued)			
no.	**description**	**variable name**	**type**
6	number of pattern/class pairs in training set	TrainSetSize	int
7	name of file containing testing set	test_file	cs
8	testing set file type: 0: pattern grouped 1: continuous	test_set_type	int
9	number of pattern/class pairs (if pattern grouped) or iterations (if continuous) in testing set	TestSetSize	int
10	name of file containing operation mode data set	operation_file	cs
11	operation mode data set file type: 0: pattern grouped	operation_set_type	int
12	number of pattern/class pairs in operation mode data set	OperationSetSize	int
13	number of pattern presentations to use in training	MaxPat	int
14	number of pattern presentations between weight updates (if zero, iteration–wise updates will be made), for more details see last paragraph of section 1.3.8	PatsPerUpdate	int
15	option to scale errors (see section 1.3.6) 0: place equal emphasis on all errors 1: place all emphasis on errors at iteration T 2: use gaussian shaped emphasis function	ScaleErrFlag	int
16	number of training epochs between consecutive evaluations of the classification rate on the test set (see section 1.3.9)	epochs_per_test_class	int
17	option for verbose version of test set classification results (see sections 1.10 and 2.5)	long_test_cls	int
18	name of file to which test set classification results should be written	test_cls	cs
19	number of training epochs between consecutive evaluations of the classification rate on the training set (see 1.3.9)	epochs_per_train_class	int
20	option for verbose version of classification results for training set (see sections 1.10 and 2.5)	long_train_cls	int
21	name of file to which training set classification results should be written	train_cls	cs
22	number of training epochs between consecutive evaluations of the mean square error for the test set (see 1.3.9)	epochs_per_test_mse	int
23	name of file to which mean square error results should be written for test set	test_mse	cs

Content and Organization of User–Options File, *user_options* (continued)			
no.	**description**	**variable name**	**type**
24	number of training epochs between consecutive evaluations of the training set mean square error (see 1.3.9)	epochs_per_train_mse	int
25	name of file to which mean square error results on the training set should be written	train_mse	cs
26	option for desired response shaping of true–class output node: (see section 1.3.5) 0: flat desired response for duration of the pattern 1: gaussian shaped desired response	desired_resp_flag	int
27	option to save the weights resulting from training 0: don't save the weights 1: dump trained weights to a file	save_wgts	int
28	name of file to which trained weights should be written	new_wt_file	cs
29	option to save outputs of the FIR network for layers 0 through L 0: don't save the outputs 1: dump outputs to a file	save_layer_flags[0] save_layer_flags[1] ... save_layer_flags[L]	int int ... int
30	option to save layer outputs during training [mode 0], testing [mode 1], and/or operation [mode 2] 0: don't save during this mode 1: save layer outputs during this mode	save_mode_flags[0] save_mode_flags[1] save_mode_flags[2]	int int int
31	output file prefix for layer outputs	output_file_prefix	cs

2.1.1 warning regarding the length of user–specified filenames

The file names which are read in from the user–options file, *user_options*, must not exceed MAXFILENAME characters in length. The value of the constant MAXFILENAME is declared in the header file fir.h. It currently has a value of 50. File names must be specified even if they will not be used.

2.2 The network–parameter file

The network–parameter file contains the specifications for the desired network architecture and training parameters. The name of the network–parameter file is read in from the user–options file, *user_options*. The contents of the network–parameter file are read in by the function Construct_Net. The number of parameters contained in the network–parameter file is dependent on the number of layers in the desired network. The content and organization of the network–parameter file is summarized in Table 2. The first column of the table gives a description of the variable, the second column states the name of the variable, and the third column states variable type.

Table 2: Content and Organization of Network–Parameter File

Description	Variable Name	Type
random number generator seed	seed	long
maximum absolute value for initial weights if assigned randomly, the initial weight values will range between –max_init_weight and +max_init_wt	max_init_wt	double
max_weight although read in, this parameter is not currently used by the software; it was originally intended to limit, during training, the absolute value of any network weight to max_weight	max_weight	double
number of layers in network this software package includes the inputs as layer 0 of the network. Thus, if a network with L layers of processing units is desired, the number of layers in the network should be specified as L+1.	p_net->numLayers	int
number of nodes in layers 0 through L of the network (note that L = p_net->numLayers – 1)	p_net->numNdsInLyr[0]	int
	p_net->numNdsInLyr[1]	int
	•••	
	p_net->numNdsInLyr[L]	int
type of transfer function to be used by nodes in layers 0 through L. 0: linear transfer function 1: hyperbolic tangent transfer function	p_net->NL_type[0]	int
	p_net->NL_type[1]	int
	•••	
	p_net->NL_type[L]	int
filter orders for nodes in layers 0 through L	p_net->filter_order[0]	int
	p_net->filter_order[1]	int
	•••	
	p_net->filter_order[L]	int
learning factors for nodes in layers 0 through L	p_net->mu[0]	double
	p_net->mu[1]	double
	•••	
	p_net->mu[L]	double
momentum factors for nodes in layers 0 through L	p_net->momentum[0]	double
	p_net->momentum[1]	double
	•••	
	p_net->momentum[L]	double

2.2.1 warnings regarding specification of network architecture

The network architecture selected by the user is somewhat constrained by the specific classification problem attempted. For example:

- the number of inputs (number of nodes in layer 0 of the network) should agree with the dimensionality of the input sequence. For example if each input sequence consists of *f* features for each of T time frames, then the user should specify the number of inputs to the network as *f*.
- the number of nodes in layers *L* and *L–1* should both be set equal to the number of possible output classes (see section 1.4.1)
- the sum of filter–orders for layers 0 through L should be one less than the number of time frames, T, included in each input sequence. This allows the observation window length of the FIR network to coincide with the input sequence length.

In addition, the user should be aware of the following:

- the existence of an input–layer, a penultimate–layer and an output–layer is always assumed; hence the number of layers in the network should always be set to a number greater than or equal to three. Note that there are no adaptable weights associated with the input units or with the output–layer units (see section 1.4.1). Hence a 3–layer network has only one–layer of adaptable weights. For most problems, a 4–layer network should be specified so as to allow for 2 layers of processing units with adaptable weights.
- the nodes in layer 0 should have linear transfer–functions
- the filter–order for the nodes in layer 0 should be 0
- the learning–factor and the momentum for nodes in layer 0 are not used, however they must be entered

2.3 data–set files

There are three possible data–set files that may be used by the program depending on the contents of the user–options file; these include: the training–set file, the testing–set file and an operation mode data set. The names of all three files along with the number of class/pattern pairs (or iterations if continuous type data set) in each of these files are specified by the user in the user–options input file. The content and organization of the data–set files are described in the following paragraph.

Each pattern grouped data file consists of a number of class/pattern pairs. Each class/pattern pair consists of a class number (indicating the true class of the associated pattern) followed by the components of the associated pattern. The class numbers are integers (see section 2.3.2); the pattern components are double.

Each continuous data file consists of a number of input sequences. The number of iterations is contained in the user options file. The location of each class in the data file is stored in an external file as classification decisions are made external to the FIR software in the case of continuous data files.

2.3.1 warning regarding the reading in of the data–files

There is nothing in the data–file to indicate either the number of class/pattern pairs or the number of components per pattern. This information is inferred from other input files as described below.

- The number of class/pattern pairs or iterations is declared in the user–options file (the declared number of class/pattern pairs (iterations) can be any integer less than or equal to the true number of class/pattern pairs (iterations) in the file).
- The number of components per pattern is inferred from the network architecture specified in the network–parameter file. Each pattern is assumed to consists of T frames of f features where T is taken to be one more than the sum of filter–orders for layers 0 through L of the network (see section 2.2.1) , and f is assumed equal to the number of network inputs. Each pattern is thus expected to have a total of fT components. The components are arranged so that the first f components correspond to the features of the first time frame, the next f components correspond to the features of the second time frame and so–on.
- The number of features, f per input vector in a continuous data set is assumed equal to the number of network inputs. The total number of components in the data file is equal to the number of iterations x f.

2.3.2 warning regarding the class numbers

The number of possible pattern classes is assumed equal to the number of nodes in the output–layer of the network (one output node is associated with each pattern class). If there are $N^{(L)}$ output nodes, then the class labels should be assigned integer values between 0 and $N^{(L)}-1$.

2.4 weight files

Weight files are both input files and output files. If the value of use_saved_weights (as read from the user–options file) is equal to 1, the program will read the initial values for the weights of the network from the file specified by the user in the user–options file; the name of this file is stored in the character array old_wt_file. If the value of save_wgts (as read from the user–options file) is equal to 1, the program will write the values of the network's weights to a file following training; the name of the file to which the weights will be written is supplied by the user in the user–options file and is stored in the character array new_wt_file. If during execution an interrupt (control c) is encountered, the current weights will be written to a file called *dumped.weights*.

The weights of the network are double precision. The order in which the weights are read from or written to a weight file is as follows. The weights associated with the first–layer of processing units are read in first, followed by the weights associated with the second–layer of processing units, ..., followed by the weights associated with the last layer of processing units. For each layer: the weights associated with the first node in the layer are read in first, followed by the weights associated with the second node in the layer, ..., followed

by the weights associated with the last node in the layer. For each node: the bias weight of the node is read in first, followed by the weights associated with the filter which connects the output of the first node on the previous layer (or first input in the case the current layer is the first layer) to the current node, followed by the weights associated with the filter which connects the output of the second node on the previous layer to the current node, ..., followed by the weights associated with the filter which connects the output of the last node on the previous layer to the current node. For each filter: the weight associated with the most current input to the filter is read in first, ..., followed by the weight associated with the oldest value stored in the filter.

2.4.1 note to user regarding nonadaptable weights

The nonadaptable weights of the last layer are also included in the weight file. The weights associated with the filters connecting corresponding nodes of the penultimate layer and final layer will all be equal to a constant (the reciprocal of the number of filter weights), all other weights associated with the last layer will be equal to 0.

2.5 classification–result files

If the value of epochs_per_test_class (epochs_per_train_class), as read in from the user–options file, is greater than zero, then the classification accuracy achieved by the network for the patterns in the test set (training set) will be evaluated every epochs_per_test_class (epochs_per_train_class) epochs of training. The classification results for the test set (training set) are written to the file specified by the user in the user–options file; the name of this file is stored in the character array test_cls (train_cls). The format of the information written to the file (following each evaluation) depends on the value of the long_test_cls (long_train_cls) as read in from the user–options file. If this value is equal to 1, then a verbose version of the classification results on the test set (training set) is produced; otherwise, a one–line summary is written. The summary of the classification results includes the number of patterns correctly classified, the number of patterns incorrectly classified, and the resulting classification rate. The verbose version of the classification results includes a line of output for every pattern in the set followed by a summary line. Each line of output generated for a particular pattern includes: the class of the pattern as decided by the network, the true class of the pattern, and a one or zero to indicate whether or not

the pattern was correctly classified[6] by the network (a one indicates correct classification). The verbose version of the classification results allows the generation of a confusion matrix; to avoid the creation of excessively long files, the verbose version should not be used in conjunction with frequent evaluation of the classification rate on the test and/or training sets. Note that the outputs of the network constitute the classification results for a continuous data set; the value of long_test_cls has no impact on the format of the results.

2.6 MSE–result files

If the value of epochs_per_test_mse (epochs_per_train_mse), as read in from the user–options file, is greater than zero, then the mean square error (MSE) between the network outputs and the desired outputs for the patterns in the test set (training set) will be evaluated (as described in section 1.11) every epochs_per_test_mse (epochs_per_train_mse) epochs of training. The MSE results for the test set (training set) are written to the file specified by the user in the user–options file; the name of this file is stored in the character array test_mse (train_mse). A single number indicating the MSE is written to the file following each evaluation. Note that the value of epochs_per_test_mse should be set to 0 when using a continuous type data set.

2.7 operation mode classification result file

During operation mode a file called *pattern_cls_summary.dat* is created; this file contains the classification decisions of the FIR network for the pattern grouped operation mode data set. Each line in *pattern_cls_summary.dat* contains the pattern class number (which varies between 0 and the number of classes – 1) that the FIR network declared present for each pattern in the data set. This file was created such that the class decision summary information could be easily extracted and graphed by the GUI.

2.8 layer output files

If save_layer_flags[ℓ] is set to a nonzero value then the outputs of all nodes in layer ℓ will be recorded for each iteration during training (if save_mode_flags[0] = 1), during testing (if save_mode_flags[1] = 1) and during operation mode (if save_mode_flags[2] = 1). The naming convention for each output file is as follows: output_file_prefix (set in *user_options*) followed by the layer number, followed by the extension *.dat*. One line is generated

6. Although this last column is redundant in that it can easily be deduced from the first two columns, it allows the user to quickly identify misclassified patterns.

for each iteration of execution and each line contains the output for the first through the last node of the chosen layer(s). Training requires $3T-2$ iterations per pattern, testing requires T iterations per pattern, and $2T-1$ iterations are performed during the operation mode for a pattern of length T iterations.

3 Using *xfir*

Before using the Graphical User Interface (GUI) for the FIR Neural Network (FIR NN), the file *xfir* (GUI executable) should be placed in the same directory as the FIR software package. The two programs are separate entities but the GUI assumes that certain files such as *fir_nn* (FIR NN executable) and its associated input and output files will be present in its directory. If *xfir* is executed outside the FIR NN software directory, the main window will appear correctly but the buttons on the display will be inoperative. The GUI is divided into four sections: Option buttons, Display buttons, *fir_nn* status display, and the quit button.

3.1 Options

The four buttons in this section all pertain to the execution of *fir_nn* and the editing of the input files (user options file and network parameter file) required to run the FIR NN software.

3.1.1 Edit user_options

Pressing this button will create another window which contains all the options and a brief description of each option in the user options file. From here the user can modify the user options file by placing the arrow in the appropriate box and making the desired change. When all the desired changes have been made, the apply button should be chosen, after which the user options file will be updated and the window will disappear. If the user wishes to abort making changes to *user_options* the cancel button can be selected and the window will disappear and *user_options* will not be updated. If the boxes in the Edit User Options window appear to be offset such that the text descriptions are no longer aligned with the input boxes then the resource file *Xfir* (located in the *include* directory) will need to be modified. Each input box has a corresponding height (i.e., XFir*opt#.height:_) and each text description an offset for the test description above it (i.e., XFir*test_op#.topOffset:_). Once these changes have been made the user must type *xrdb Xfir* in the *include* directory to implement the changes. Some trial and error may be required before the desired window sizing is achieved as the parameters may vary according to the operating platform. If no user options file is present (in the current directory) all the input boxes will appear empty, when all the fields have been entered and the apply button is selected then a file *user_options* will be created. The Edit User Options button is always available for selection (*i.e.*, will never be greyed out except when window is displayed) but changes can only be acted upon if they are made before the simulation is begun.

3.1.2 Edit Net Parameters

Pressing this button will cause another window called the Edit Network Parameters to appear. The contents of the network parameter file, whose name is listed in the user options file, should be displayed in the input boxes alongside a short description and list of the available options for each input to the network parameter file. When the user has made all the desired changes, the apply button can be selected and the network parameter file will be updated and the Edit Network Parameter window will disappear. If the user wishes to abort making changes to the network parameter file then the cancel button can be pressed and the Edit Network Parameter window will disappear and the network parameter file will remain unchanged. If no network parameter file is present then the input boxes will appear empty; when the user has finished entering the parameters and has selected the apply button then a network parameter file whose name is listed in the user options file will be created. Like the Edit user_options button, the Edit Net Parameters button is always available unless already chosen but new changes can only be acted upon if they were made before the simulation was begun.

3.1.3 Begin Simulation

Pressing the Begin Simulation button will issue a command to execute the *fir_nn* program, if *fir_nn* is not in the same directory as *xfir* when the simulation is begun the GUI will not be able to start the *fir_nn* program rendering the GUI essentially useless. Assuming that all the executables and the input files are in their proper directory and the input files are in the proper format then when the Begin Simulation button is selected the *fir_nn* program will be executed according to the instructions contained in the user options file. Shortly after the simulation is begun the status display should switch from idle to some other state and two of the display options may become available depending on the configuration of the user options file. After the simulation has begun, the Begin Simulation button remains unavailable until the *xfir* program is terminated; therefore each time a separate simulation is to be tried the GUI must be re–executed.

3.1.4 End Simulation

Once the simulation has begun the user may wish to terminate execution of *fir_nn* prematurely for a variety of reasons (*e.g.*, the performance of the network may have reached an acceptable level, the user may wish to modify the user options file or the network parameter file, etc.). The End Simulation button only becomes available for selection when the simulation has begun. When the simulation is terminated by use of the End Simulation button, the current weight values will be saved to a file called *dumped.weights* to prevent any loss of training. If *dumped.weights* exists previously to pressing the End Simulation button then its old contents will be over written with the current weight values of the FIR network.

3.2 Display

There are four buttons in this category, two of which are operational (Network Outputs and Class Decision Summary) and two which are currently unavailable (Classification Rate and Mean Square Error). All options display output files generated from the FIR NN program.

3.2.1 Network Outputs

Once the FIR NN simulation has begun, the Network Outputs window will appear with the option of selecting the layer, graph type, and sweep time. The sweep time, maximum number of samples displayed, is set to 1024 and can be changed if the user wishes to do so. To select another sweep time the user must select the sweep time button. The Sweep Time window will then appear, that user has the choice of choosing between four preset values 200, 500, 800, and 1024 or they may enter a value of their choice. Any value between the minimum sweep time of 200 and maximum sweep time of 1024 is valid, any value outside this range will get set to the appropriate outer limit. When the user has made the change and selected the apply button the new value will be displayed on the sweep time button. The user can also choose between two types of visual display (waterfall and brightness) for the graphing of the node outputs. Waterfall graphs are xy plots with the node output plotted versus iteration number. Brightness graphs plot node output (represented by different color bars) versus iteration number. The node outputs will vary over a scale of -1 to $+1$ with the color scale changing from black to red to yellow to white, where -1 is represented by black, 0 by red, and $+1$ by white. The user also has the option to display the node outputs of the layers that were specified in line 29 of the user options file. For each layer selected on this line a separate output file will be created and the GUI will have the option to display the data contained within each of these files. Once the user has selected a layer another window will appear listing all the possible nodes of that layer, the user may select any of these nodes. Each node will have its own window showing the node number, the layer number, and K_0 the index referring to the actual iteration index for the leftmost edge of the node output graph. The graphs will keep updating until the end of the data file has been reached.

3.2.2 Class Decision Summary

Another graph which the user has the choice of viewing is a classification decision summary graph. This option only becomes available when a data set is processed during operation mode (see sections 1.3.11 and 2.7) where the weights are held constant and the patterns are clocked in and out of the FIR network, this option is chosen by placing a 1 in the third column of line 30 of the user options file. The classification decision is based on choosing the node with highest output when the entire pattern is clocked into the network, this node number is then written to the output file *pattern_cl_summary.dat*. The summary information is then extracted from this file. The

window Pattern Based Class Decision Summary allows the user to select the beginning and end of the summary interval with the defaults being the start and end of the current contents of *pattern_cls_summary.dat* or the user may select any range between the existing defaults. When the user has selected the desired summary interval and selected the summarize button a decision summary graph will be produced. The display will show a bar graph with class category along the horizontal axis and number of occurrences along the vertical axis. The summary is a static display and will not update as new patterns are processed, to view a second summary simply repeat the process and select the appropriate interval.

3.3 *fir_nn* Status Display

The Status display is meant to give the user a general idea of where the FIR NN program is within its execution cycle. The possible displays are:

- Idle
- Reading: <filename>
- Training: <filename>
- Classifying: <filename>
- Computing MSE: <filename>
- Operation: <filename>

where <filename> is the name of the data file on which the action is being performed. The FIR NN program writes to a file called *fir_nn.status* each time a new action is begun. The GUI program scans this file and displays its contents, to save processing time the scan interval was set to one second, actions which require less than one second to be performed may not be displayed, unless the data sets are quite small this should not be a problem.

3.4 Quit

We suggest that the users always choose the quit or cancel buttons which appear as a button within the window as opposed to one that might appear on the window border as part of the window manager system; this will ensure a clean termination of the window.

References

[1] Arsenault, N. and M. Stevenson, *Classification of Simulated Sequences using an FIR Neural Network: Experimental Results.*, CR/97/459, April 1995.

[2] Stevenson, M., *Survey of Dynamic Neural Network Techniques with Application to Temporal Processing Tasks*, CR/97/458, April 1995.

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)
University of New Brunswick
P.O. Box 440, Fredricton, N.B., Canada
E3B 5A3

2. SECURITY CLASSIFICATION (Overall security of the document including special warning terms if applicable.)
UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title.)
FIR Neural Network Software Package User's Manual

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)
STEVENSON, Maryhelen and ARSENAULT, Normand

5. DATE OF PUBLICATION (Month and year of publication of document.)
April 1995

6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)
31

6b. NO. OF REFS. (Total cited in document.)
2

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)
Contractor Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development. Include the address.)
Defence Research Establishment Atlantic
P.O. Box 1012, Dartmouth, N.S., Canada B2Y 3Z7

9a. PROJECT OR GRANT NUMBER (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)
1cc

9b. CONTRACT NUMBER (If appropriate, the applicable number under which the document was written.)
W7707-3-2682/01-OSC

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

10b. OTHER DOCUMENT NUMBERS (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DREA Contractor Report 97/460

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; futher distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic annoucement of this document. This will normally correspond to the Document Availability (11). However, where futher distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

DCDCD 2/06/87

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

User's manual (revised edition, April 1995) for Finite Impulse Response Neural Network software written in the C programming language under the contract. The revised version includes the operation of a Graphic User Interface (GUI), and is updated to reflect all the changes in the use and operation of the FIR Neural Network and the GUI software packages.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

passive sonar
neural networks
dynamic neural networks
finite impulse response
transient signals